1. Giới thiệu Express:
   1. Express là gì ?
      * Express js là một Framework nhỏ, nhưng linh hoạt được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile
      * Về các package hỗ trợ: Expressjs có vô số các package hỗ trợ nên các bạn không phải lo lắng khi làm việc với Framework này.
      * Về performance: Express cung cấp thêm về các tính năng (feature) để dev lập trình tốt hơn. Chứ không làm giảm tốc độ của NodeJS.
      * Và hơn hết, các Framework nổi tiếng của NodeJS hiện nay đều sử dụng ExpressJS như một core function, chẳng hạn: SailsJS, MEAN,…
   2. Hướng dẫn cài đặt:
      * Do Express framework không được tích hợp sẵn vào trong node.js nên khi muốn sử dụng nó chúng ta phải tải nó về máy với sử dụng được. Và để cải đặt nó chúng ta sử dụng cú pháp của npm: npm install express –save
   3. Các Route trong Express:
      * Route GET method: Router này chỉ nhận phương thức get đến url.

**VD**: trả về chữ 'hello world' khi người dùng get request vào url có param là say.

app.get(‘/say’, function(req,res){

res.send(‘hello word’);

});

* + - Route POST method: Router này chỉ nhận phương thức post.

**VD**: Trả về chữ 'hello world' khi người dùng gửi post request đến url có param là say.

app.post(‘/say’, function(req,res){

res.send(‘hello word’);

});

* + - Các Method khác: Đối với các method khác cung làm tương tự như hai method get và post. Chỉ việc đổi tên method thành method mà bạn muốn nhận.

//put method

app.put(‘/say’, function(req,res){

res.send(‘hello word’);

});

//delete method

app.delete (‘/say’, function(req,res){

res.send(‘hello word’);

});

* + - Nhận tất cả các Method: Trong trường hợp các bạn muốn nhận tất cả các method thì chỉ cần đổi method thành all là ứng dụng sẽ nhận tất cả các method.

app.all(‘/say’, function(req,res){

res.send(‘hello word’);

});

* + - Truyền tham số vào router: Để truyền tham số vào router chúng ta sẽ sử dụng dấu : và tên biến vào path.

app.get(‘/user/:name, function(req,res){

res.send(‘User có tên là: ‘ + req.params.name);

});

* 1. Views and Layouts: Trong Express hỗ trợ nhiều Template engine dùng cho việc tạo Views nhanh chóng và cần thiết cho app

var app = express();

// set up template engine

app.set(‘view engine’, ‘ejs’);

app.set(‘views’, ‘./Views/page’);

* 1. Static Files and Views: Các tệp và dạng xem tĩnh Express dựa trên một phần mềm trung gian để xử lý các tệp và chế độ xem tĩnh. Phần mềm trung gian tĩnh cho phép bạn chỉ định một hoặc nhiều thư mục chứa tài nguyên tĩnh đơn giản được gửi tới máy khách mà không cần xử lý đặc biệt. Đây là nơi bạn sẽ đặt những thứ như hình ảnh, tệp CSS và tệp JavaScript phía máy khách. Trong thư mục dự án của bạn, tạo một thư mục con được gọi là public (chúng ta gọi nó là public bởi vì bất cứ thứ gì trong thư mục này sẽ được phục vụ cho client). Sau đó, trước khi bạn khai báo bất kỳ tuyến nào, bạn sẽ thêm phần mềm trung gian tĩnh:

Khai báo:

app.use(express.static(\_dirname + ‘/public’));

vd:

<img src=”/img/logo.png”>

* 1. Ngoài ra: nó còn giúp chúng ta truyền được những dữ liệu vào Views một cách dễ dàng:

Phần view: about.ejs

<%= name %>

Phần controler:

app.get(‘/’, function(req,res){

res.render(‘about’, {name: “Hello”});

});

## Thành phần của một đường dẫn URL

Đây là các thành phần chính của URL này :

* Giao thức là http. Ngoài ra còn có nhiều giao thức khác như https, ftp, v.v;
* Tên host hay hostname là video.google.co.uk;
* Tên miền thứ cấp là video;
* Tên miền chính là google.co.uk;
* Tên miên cấp 1 (top-level domain) la uk.Tên miền uk là tên miền quốc gia
* Tên miền cấp 2 (second-level domain) là co.uk;
* Cổng giao tiếp là 80, đây là cổng ngầm định của dịch vụ Web. Có thể sử dụng các cổng khác như 8080 hay 8000. Khi cổng là 80, phần lớn người ta không cần thêm vào đường dẫn;
* Đường dẫn là /videoplay, giúp để trỏ đến một tệp tin hay một vị trí nằm trên máy chủ Web;
* Đường dẫn URL nói trên ẩn chứa các tham số. Tên của tham số là docid và giá trị của nó là -7246927612831078230. Đường dẫn có thể có nhiều tham số. Tham số đầu tiên bắt đầu bằng dấu hỏi (?), các tham số sau nối tiếp bằng dấu và (&);
* Ký tự “#00h02m30s” còn được gọi là neo dùng để trỏ đến một vị trí trên văn bản hiện tại. Trong trường hợp trên vị trí neo trỏ đến đoạn phim thời điểm 2 phút, 30 giây trong đoạn video tương ứng.

1. The Request and Response Objects:

Nếu như lập trình front-end quan tâm đến giao diện, đến trải nghiệm người dùng, đến HTML, CSS và JavaScript, thì lập trình back-end quan tâm đến việc chọn lọc và gửi đi dữ liệu chứa trong database phù hợp với yêu cầu của người dùng.

Do phía server liên tục:

* nhận các HTTP request từ trình duyệt,
* xử lý các HTTP request
* lôi dữ liệu từ database ra xử lý theo yêu cầu
* tạo HTTP response để phản hồi cho trình duyệt

cho nên có 3 đối tượng chính mà lập trình back-end cần học để làm việc:

1. HTTP request
2. HTTP response
3. Data trong database

HTTP resquest :

* HTTP là viết tắt của 1 giao thức, tạm gọi đây là quy định về giao tiếp giữa server với trình duyệt.
* request là yêu cầu. Tức là 1 yêu cầu của trình duyệt đòi server gửi thông tin để render 1 trang web, hoặc gửi 1 file nhạc, file ảnh.

Đã là giao thức, thì bao giờ cũng phải có các tiêu chuẩn quy định , có thể thì mới giúp 2 bên hiểu nhau trong nhiều tình huống, không bị hiểu nhầm, hiểu sai.

Trong mỗi HTTP request có gì? Có 1 vài thứ:

* có URL để trỏ đến tài nguyên trên web server mà trình duyệt đang cần. Tài nguyên là gì? file HTML, JS, CSS, hoặc thậm chí 1 file JSON.
* bản chất của HTTP request có vài loại, như GET, POST, PUT, UDPATE, DELETE, HEAD. Mỗi cái tương ứng với 1 loại công việc mà trình duyệt yêu cầu web server đáp ứng.
* data được encoded bên trong body của HTTP request.

HTTP request thực chất là 1 file text. Tùy vào từng loại HTTP request mà file text khác nhau ít nhiều. Tuy vậy, trong thực tế, người ta dùng dạng POST trong trường hợp người dùng submit form để gửi thông tin đi, do dạng POST encode thông tin vào body của request thay vì để nó ra ngoài URL giống như GET

### HTTP response

HTTP response thì mang trong mình nó 2 thứ:

* Code status, để báo hiệu cho trình duyệt biết là request nó gửi đi được xử lý thành công hay không.
* Dữ liệu mà trình duyệt cần.

Template Engine

Có rất nhiều template engine được sử dụng cho Nodejs. Nếu chúng ta sử dụng Express.js thì có thể cái plug-in consolidate.js nó hỗ trợ rất nhiều template engine, có thể kể ra danh sách template engine như sau.

* atpl
* dust (website)
* eco
* ect (website)
* ejs
* haml (website)
* haml-coffee (website)
* handlebars (website)
* hogan (website)
* jade (website)
* jazz
* jqtpl (website)
* JUST
* liquor
* mustache
* QEJS
* swig (website)
* templayed
* toffee
* underscore (website)
* walrus (website)
* whiskers

Nhưng với việc có rất nhiều template engine thì sẽ khiến chúng ta phân vân không biết nên chọn cái nào. Để trả lời cho câu hỏi cái nào sẽ là template engine tốt nhất cho Nodejs? Câu trả lời có lẽ là không có, vì nếu đã có một template engine tốt nhất rồi thì tại sao lại còn có những template engine khác làm gì. Ở đây chúng ta chỉ có thể trả lời cho việc Template engine nào được sử dụng nhiều nhất, và điểm mạnh, điểm yếu của mỗi cái để người dùng có thể chọn lựa sao cho phù hợp với dự án.

# Những Template Engine phổ biến

Theo như nghiên cứu tài liệu từ một số chỗ, chẳng hạn như Jade có vẻ như là template engine được dùng phổ biến nhất, nó cũng là template engine mặc định khi mà cài đặt Express.js, nhưng có vẻ như khá nhiều khá thích sử dụng EJS bởi vì tính thuận tiện và đơn giản

# EJS

Nếu đã quen thuộc với việc sử dụng HTML thì việc dùng EJS khá dễ dàng, vì với syntax khá giống với html, bạn có thể tạo ra các partial view với ejs để giúp cho việc reused trở nên dễ dàng và giúp cho code dễ đọc, dễ bảo trì hơn. EJS giúp bạn truyền data vào các views. Một số bước để có sử dụng EJS với Nodejs

## Cài đặt

chỉ cần chạy lệnh

$ npm install ejs

## Sử dụng

var express = require('express');

var app = express();

app.set('view engine', 'ejs');

app.get('/', function(req, res){

res.render('index',{user: "Great User",title:"homepage"});

});

Chúng ta cần bước set view Engine cho ejs

app.set('view engine', 'ejs');

vì mặc định khi cái express.js thì default view engine sẽ là Jade(pug) Chúng ta truyền view tới user bằng cú pháp

res.render('index',{user: "Great User",title:"homepage"});

với index là view và dữ liệu được truyền là user và title. Với việc tạo và sử dụng Partial views giả sử chúng ta đã tạo ra 2 file là header.ejs

<title>

Hello World

</title>

và body.ejs

<div>

Welcome, Great User

</div>

thì ở chỗ gọi tới 2 file đó chúng ta dùng từ khóa include

<html>

<head>

<% include('header.ejs') %>

</head>

<body>

<% include('body.ejs') %>

</body>

</html>

Sử dụng javascript cho tempalating trong ejs Ở trong ejs không có hỗ trợ block, mà những việc thực hiện với các danh sách chúng ta phải sử dụng forEach

<ul>

<% users.forEach(function(user){ %>

<%= user.name %>

<% })%>

</ul>

Với data tương ứng cho danh sách như sau

app.get('/', function(req, res){

res.render('index',{users : [

{ name: 'name1' },

{ name: 'name2' },

{ name: 'name3' }

]});

});

FORM HANDLING

Tổng quan:

HTML FORM là một nhóm gồm nhiều trường trên một trang web dùng để thu thập thông tin người dùng gửi lên máy chủ. Form rất linh hoạt trong việc thu thập dữ liệu người dùng vì đầu vào của nó thu thập được nhiều dữ liệu khác nhau như: text boxes, checkboxes, radio buttons, date pickers, … Với phương thức POST dữ liệu được gửi lên máy chủ một cách an toàn.

Tổng quan của một HTML Form gồm một ô nhập liệu người dùng và nút submit để gửi lên máy chủ. Khi người dùng đã hoàn tất việc nhập liệu và gửi lên máy chủ thì Form sẽ tìm đến thẻ <form></form> để xác định phương thức trong form và dữ liệu người dùng. Nếu không xác định phương thức cho form thì mặc định, phương thức sẽ là GET

<form action="/team\_name\_url/" method="post">

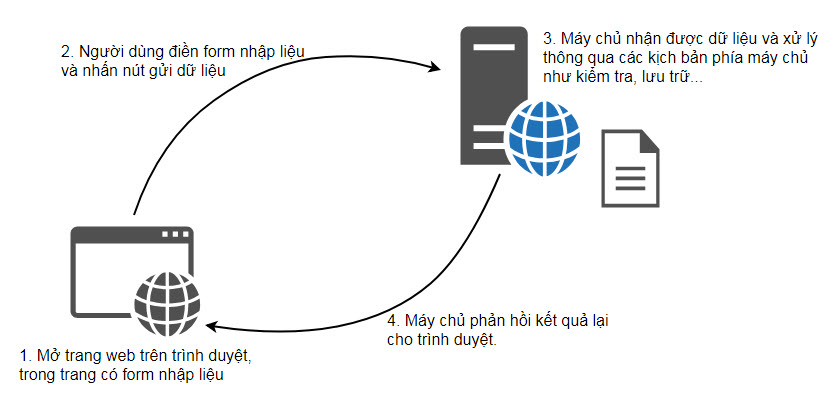
<label for="team\_name">Enter name: </label>

<input id="team\_name" type="text" name="name\_field" value="Default name for team.">

<input type="submit" value="OK">

</form>

Tiến trình của Form được hoạt động như sau:



1. Người dùng mở trang web có form nhập liệu trong trình duyệt.  
2. Người dùng điền thông tin vào các thành phần form nhập liệu sau đó nhấn nút gửi (submit) dữ liệu lên máy chủ. Trong giai đoạn này dữ liệu được nhập vào đã có thể được kiểm tra tại máy tính người dùng thông qua các đoạn mã Javascript.  
3. Tại máy chủ web, thông tin được xử lý thông qua các ngôn ngữ kịch bản máy chủ như PHP, ASP.NET, JSP… Các đoạn mã kịch bản máy chủ sẽ thực hiện kiểm tra dữ liệu trong form, thực hiện các logic, ghi nhận dữ liệu vào database…  
4. Máy chủ web sau khi xử lý xong thông tin sẽ gửi lại một hồi đáp thành công hay không về trình duyệt và kết thúc một chu trình khép kín xử lý form nhập liệu.

Form handling with Express:

Nếu sử dụng phương thức GET cho Form, thì sẽ được gọi bằng: req.query.object

Còn nếu sử dụng phương thức POST cho Form, thì sẽ cần đến đường dẫn trong Middleware để tương tác với người dùng, gọi trường bằng: req.body.object

Session and Cookie:

# Cookie

### Cookie là gì?

**Cookie** là một đoạn văn bản ghi thông tin được tạo ra và lưu trên trình duyệt của máy người dùng. **Cookie** thường được tạo ra khi người dùng truy cập một website, **cookie** sẽ ghi nhớ những thông tin như tên đăng nhập, mật khẩu, các tuỳ chọn do người dùng lựa chọn đi kèm. Các thông tin này được lưu trong máy tính để nhận biết người dùng khi truy cập vào một trang web.

Khi người dùng truy cập đến một trang web có sử dụng **cookie**, web server của trang đó sẽ tự động gửi **cookie** đến máy tính của người dùng. Những **cookie** này tự động được tổ chức trong hệ thống máy tính. Khi truy cập đến các trang web sử dụng được **cookie** đã lưu, những **cookie** này tự động gửi thông tin của người dùng về cho chủ của nó (người tạo ra **cookie**). Tuy nhiên những thông tin do **cookie** ghi nhận không được tiết lộ rộng rãi, chỉ có website chứa **cookie** mới có thể xem được những thông tin này. **Cookie** được xem là một thành phần không thể thiếu được với những website có khối lượng dữ liệu lớn, có số lượng người dùng đông, và có những chức năng đi kèm với thành viên đăng ký. Phần lớn các website này là các website thương mại điện tử.

# Session

### Session là gì?

**Session** là một khái niệm phổ biến được dùng trong lập trình các website có kết nối với cơ sở dữ liệu database. Đặc biệt các chức năng như đăng nhập, đăng xuất người dùng sẽ khó có thể thực hiện được nếu không sử dụng **session**.

**Session** đơn giản là 1 cách để chúng ta lưu lại dữ liệu của người dùng sử dụng website. Giá trị của **session** được lưu trong một tập tin trên máy chủ. Ví dụ khi bạn đăng nhập vào một trang web và đăng nhập với tài khoản đã đăng ký trước đó. Máy chủ sau khi xác thực được thông tin bạn cung cấp là đúng nó sẽ sinh ra một tập tin (hay chính là **session** của trình duyệt của bạn) chứa dữ liệu cần lưu trữ của người dùng.

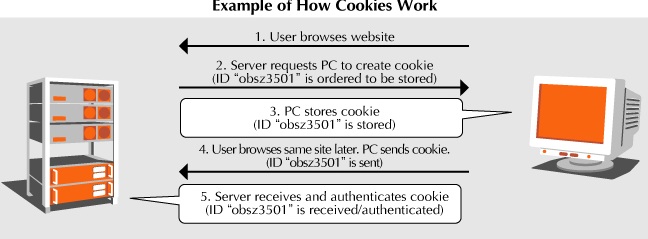
Bạn có thể tuỳ ý quyết định xem nên lưu trữ những thông tin nào vào Session. Nhưng thông thường chúng ta chỉ nên lưu những thông tin tạm thời trong **session** ví dụ như số lượng sản phẩm người dùng đã thêm vào giỏ nhưng chưa mua, hay những nhật xét đang được viết và lưu dưới dạng nháp nhưng chưa gửi đi. Những dữ liệu sử dụng lâu dài như nội dung nhận xét đã được gửi đi hay số sản phẩm đã được mua thì nên được thực hiện ở máy chủ chứa cơ sở dữ liệu.

### Cách phân biệt session của các trình duyệt khác nhau

Sau khi tạo ra một tập tinh **session** trên máy chủ để lưu trữ dữ liệu tạm thời của người dùng, chúng ta cần phải phân biệt được **session** nào là của người dùng nào. Để làm điều này thì với mỗi **session** tạo ra cần phải tạo một **cookie** tương ứng với nó. **Cookie** là một mẫu tin nhỏ có thể được trình duyệt tạo ra khi người dùng truy cập vào web và dùng để lưu trữ thông tin của người dùng ở phía trình duyệt (client).

# So sánh giữa Cookie và Session

| **Cookie** | **Session** |
| --- | --- |
| **Cookie** được lưu trữ trên trình duyệt của người dùng. | **Session** không được lưu trữ trong trình duyệt của người dùng. |
| Dữ liệu **cookie** được lưu trữ ở phía máy khách. | Dữ liệu **session** được lưu trữ ở phía máy chủ. |
| Dữ liệu **cookie** dễ dàng sửa đổi khi chúng được lưu trữ ở phía khách hàng. | Dữ liệu **session** không dễ dàng sửa đổi vì chúng được lưu trữ ở phía máy chủ. |
| Dữ liệu **cookie** có sẵn trong trình duyệt của chúng ta đến khi hết hạn. | Dữ liệu **session** có sẵn cho trình duyệt chạy. Sau khi đóng trình duyệt sẽ mất thông tin **session**. |



Nodejs là ngôn ngữ bất đồng bộ, bản thân nó không cung cấp các phương thức set/get session như những ngôn ngữ khác. Mặc dù nó có cung cấp cơ chế lấy cookie trong module http nhưng việc truy xuất cũng vô cùng nhiều khó khăn.

Nhưng có một module giúp bạn làm việc này là express-session. Cơ chế của nó không khác những hệ thống khác, vẫn là dùng một cookie ở client để đánh dấu ID client, và data session được lưu tại server. Để dùng module express-session bạn chỉ cần load module này về bằng câu lệnh

npm install express-session

var session = require(‘express-session‘)

Sau đó chỉ việc gắn nó vào hệ thống

var app = express()

app.set(‘trust proxy’, 1) // trust first proxy

app.use(session({

secret: ‘'secret'’,

resave: true,

saveUninitialized: true,

cookie: { secure: true }

}))

Có một điểm bất tiện, default của nó là lưu thông tin session trong memory. Điều này có nghĩ là nếu phải restart lại node thì coi như mất hết. Để khác phục điều này, ta có thể cho lưu các giá trị session vào database, vừa an toàn khi lưu trữ, vừa tiện việc đồng bộ dữ liệu. Có nhiều module để lưu session value vào các database system khác nhau như MongoDB, PostgreSQL, MySQL, MariaDB, SQLite3, Oracle . . .

Vd: Để lưu session vào database, thì có thể dùng package: ['connect-mongo'];

var MongoStore = require('connect-mongo')(session);

app.use(session({

secret: 'secret',

resave: true,

saveUninitialized: true,

store: new MongoStore({

mongooseConnection: mongoose.connection

}),

cookie: {

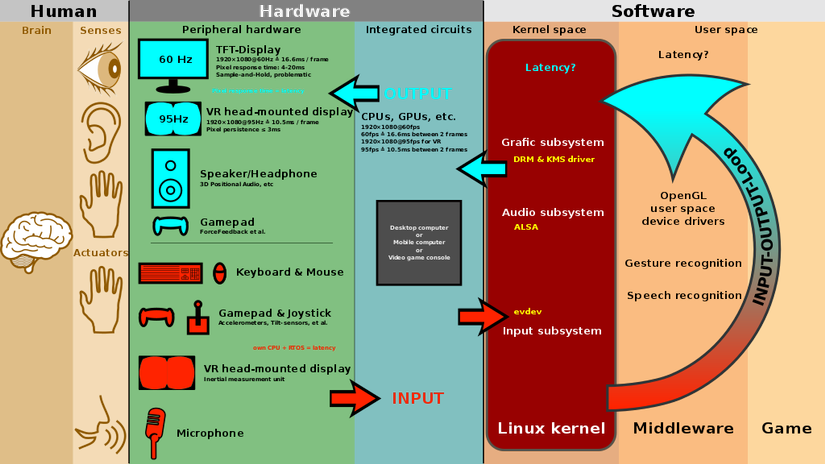
maxAge: 180\*60\*1000

}

}));

Middleware:

Middleware trong ngành công nghệ phần mềm được định nghĩa là một phần mềm có nhiệm vụ làm cầu nối (bridge), cung cấp các dịch vụ từ phía hệ điều hành đến các ứng dụng, giúp các ứng dụng có thể tương tác với các thành phần được hệ điều hành cho phép. Middleware được coi là chất kết dính dữa các phần mềm với nhau.

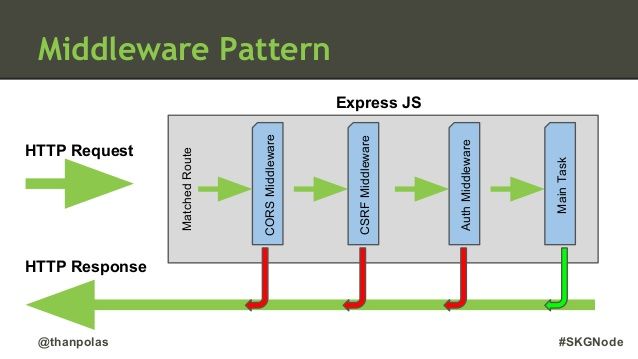


Trong hình, đối với một hệ điều hành (VD Linux), Middleware sẽ đóng vai trò là các phần mềm nằm giữa các ứng dụng tương tác trực tiếp với người dùng và nhân - kernel của hệ điều hành. Đó có thể là thư viện xử lý hình ảnh OpenGL, Driver các thiết bị phần cứng/mềm, phần mềm nhận dạng giọng nói / chữ viết...

Middleware trong Web

Với tư tưởng chung là cầu nối giữa tương tác của người dùng và phần nhân của hệ thống, trong lập trình Web, Middleware sẽ đóng vai trò trung gian giữa request/response (tương tác với người dùng) và các xử lý logic bên trong web server.

Do đó, Middleware trong các Framework lập tình Web (Django, Rails, ExpressJS), sẽ là các hàm được dùng để tiền xử lý, lọc các request trước khi đưa vào xử lý logic hoặc điều chỉnh các response trước khi gửi về cho người dùng.



Hình trên mô tả 3 middleware có trong ExpressJS. Một request khi gửi đến Express sẽ được xử lý qua 5 bước như sau :

1 Tìm Route tương ứng với request

2 Dùng CORS Middleware để kiểm tra cross-origin Resource sharing của request

3 Dùng CRSF Middleware để xác thực CSRF của request, chống fake request

4 Dùng Auth Middleware để xác thực request có được truy cập hay không

5 Xử lý công việc được yêu cầu bởi request (Main Task)

Bất kỳ bước nào trong các bước 2,3,4 nếu xảy ra lỗi sẽ trả về response thông báo cho người dùng, có thể là lỗi CORS, lỗi CSRF hay lỗi auth tùy thuộc vào request bị dừng ở bước nào.

Middleware trong ExpressJS

ExpressJS khi hoạt động, về cơ bản sẽ là một loạt các hàm Middleware được thực hiện liên tiếp nhau. Sau khi đã thiết lập, các request từ phía người dùng khi gửi lên ExpressJS sẽ thực hiện lần lượt qua các hàm Middleware cho đến khi trả về response cho người dùng. Các hàm này sẽ được quyền truy cập đến các đối tượng đại diện cho Request - req, Response - res, hàm Middleware tiếp theo - next, và đối tượng lỗi - err nếu cần thiết.

Một hàm Middleware sau khi hoạt động xong, nếu chưa phải là cuối cùng trong chuỗi các hàm cần thực hiện, sẽ cần gọi lệnh next() để chuyển sang hàm tiếp theo, bằng không xử lý sẽ bị treo tại hàm đó.

Các chức năng mà middleware có thể thực hiện trong ExpressJS sẽ bao gồm :

Thực hiện bất cứ đoạn code nào

Thay đổi các đối tượng request và response

Kết thúc một quá trình request-response

Gọi hàm middleware tiếp theo trong stack

Trong Express, có 5 kiểu middleware có thể sử dụng :

Application-level middleware (middleware cấp ứng dụng)

Router-level middleware (middlware cấp điều hướng - router)

Error-handling middleware (middleware xử lý lỗi)

Built-in middleware (middleware sẵn có)

Third-party middleware (middleware của bên thứ ba)

Application-level middleware

Khi khởi tạo một Web Application với ExpressJS, chúng ta sẽ có một đối tượng đại diện cho Web App đó, thường được gán với biến app. Đối tượng này có thể khai báo các middleware thông qua các hàm : app.use() hoặc app.METHOD (trong đó METHOD sẽ là cá kiểu HTTP Method được ExpressJS hỗ trợ, dưới dạng tên là chữ viết thường, vd app.get(), app.post()).

Ví dụ dưới đây mô tả một hàm ko khai báo đường dẫn cụ thể, do đó nó sẽ được thực hiện mỗi lần request:

var app = express()

app.use(function (req, res, next) {

console.log('Time:', Date.now())

next()

})

Router-level middleware

Các middleware này về chức năng không khác gì so với application-level middlewware ở trên, tuy nhiên thay vì dùng biến app có thể gây nhầm lẫn với các thiết lập, phần router có thể không rõ ràng và khó phân biệt, ExpressJS cung cấp một đối tượng router chuyên dùng để khai báo route bằng cách gọi hàm sau:

var router = express.Router()

Error-handling middleware

Đây là các middleware phục vụ cho việc xử lý lỗi. Một lưu ý là các hàm cho việc này luôn nhận bốn tham số (err, req, res, next). Khi muốn khai báo một middlware cho việc xử lý lỗi, bạn cần tạo một hàm có 4 tham số đầu vào. Mặc dù bạn có thể không cần sử dụng đối tượng next, nhưng hàm vẫn cần format với bốn tham số như vậy. Nếu không ExpressJS sẽ không thể xác định đó là hàm xử lý lỗi, và sẽ không chạy khi có lỗi xảy ra, chỉ hoạt động giống như các hàm middlware khác.

Đoạn code dưới đây mô tả một hàm xử lý lỗi truyền về cho client lỗi 500 khi có lỗi xảy ra từ server:

app.use(function (err, req, res, next) {

console.error(err.stack)

res.status(500).send('Something broke!')

})

Built-in middleware

Kể từ phiên bản 4.x, ExpressJS đã không còn phụ thuộc vào thư viện Connect. Ngoài middleware express.static, tất cả các hàm middleware khác đều đã được tách ra thành các modules riêng biệt. Điều này cung cấp cách tối ưu hóa và tùy chỉnh ứng dụng ExpressJS một cách linh hoạt nhất, giúp bạn tạo ra một ứng dụng Web Application phù hợp với nhu cầu, không bị thừa những thứ không cần thiết. Bạn có thể tham khảo các modules middlware đã được tách ra ở đây

Chỉ có một Built-in middlware duy nhất còn lại trong ExpressJS là express.static, dựa trên thư viện serve-static, được dùng để cung cấp các nội dung tĩnh trong trang Web, ví dụ như các trang HTML tĩnh, các file hình ảnh, css, js, ...

Đoạn dưới đây mô tả việc sử dụng express.static để tạo ra một thư mục có tên là public, người dùng có thể truy cập các file html và htm trong thư mục này:

var options = {

dotfiles: 'ignore',

etag: false,

extensions: ['htm', 'html'],

index: false,

maxAge: '1d',

redirect: false,

setHeaders: function (res, path, stat) {

res.set('x-timestamp', Date.now())

}

}

app.use(express.static('public', options))

Ngoài ra bạn có thể khai báo nhiều thư mục static trong một web, đoạn code sau sẽ tạo ra 3 thư mục static :

app.use(express.static('public'))

app.use(express.static('uploads'))

app.use(express.static('files'))

Third-party middleware

Sử dụng Third-party sẽ giúp chúng ta thêm các chức năng cho Web App của mình mà không cần mất nhiều công implement.

Chúng ta sẽ cần cài đặt module thông qua npm, sau đó khai báo sử dụng trong đối tượng app nếu dùng ở Application-level, hoặc qua đối tượng router nếu dùng ở Router-level.

Đoạn code sẽ cài đặt và sử dụng một middlware có tên là cookie-parser dùng để đọc cookies của request:

$ npm install cookie-parser

var express = require('express')

var app = express()

var cookieParser = require('cookie-parser')

// load the cookie-parsing middleware

app.use(cookieParser())

Routing:

Routing trong Node.js là một khái niệm nói đến việc xác định ứng dụng sẽ đáp ứng như thế nào khi người dùng tạo một request đến một endpoint (Điểm cuối) cụ thể nào đó. Điểm cuối đó thường là một URI hoặc một đường dẫn (Path) với một Request method (POST, PUT, GET, ...) cụ thể.

Cấu trúc định tuyến cơ bản

Trong express.js định tuyến có cấu trúc như sau

app.METHOD(Path, Handler...)

Trong đó:

app : là một instance của express

METHOD: là một HTTP Method

Path: là một đường dẫn trên máy chủ

Handler : là một function sẽ thực thi khi một route được trùng khớp

Giải thích

handler : có thể có một hoặc nhiều function

một route được xác định bằng Path (đường dẫn) và request method.

Khái niệm route trùng khớp là chỉ việc một người dùng thực hiện request với Path (đường dẫn) và Method trùng khớp với định nghĩa trong route.

Ví dụ ra có một route như sau

app.get('/hello', function doHello(req, res) {

res.send('Hello World!')

})

Thì khi đó nếu client thực hiện một GET /hello đến máy chủ, thì khi ấy route sẽ trùng khớp và function doHello sẽ được gọi thực hiện.

Route paths (Đường dẫn)

Route path có thể là một chuỗi thông thường (String) hoặc là một chuỗi có ký hiệu biểu thức chính quy (string patterns) hoặc là một biểu thức chính quy (regular expressions). Ví dụ

/users/nghuuquyen : là một đường dẫn thông thường

/users/user/\* : là một đường dẫn với ký hiệu \* biểu diễn cho một chuỗi bất kỳ

/^[a-zA-Z0-9]{5,15}$/ : là một đường dẫn có dạng biểu thức chính.

app.get('/users/nghuuquyen', function(req, res) {

//Do something.

});

là một đường dẫn thông thường.

app.get('/users/\*', function(req, res) {

// Do something.

});

là một đường dẫn theo khuôn mẫu (String pattern). Ký hiệu \* ở đây nói lên là route trên khớp với mọi đường dẫn bắt đầu với /users/.

app.get(/.\*cool$/, function(req, res) {

// Do something.

});

là một đường dẫn với dạng biểu thức chính quy (regular expression). Route này sẽ khớp với mọi đường dẫn kết thúc với đuôi là cool. Ví dụ YOUARESOcool sẽ khớp nhưng YOUTOOCool thì sẽ không vì khác ký tự 'C'.

Ứng dụng

Kiểu string pattern mình hay áp dụng để việc đánh chặn tất cả các route, dùng trong việc bảo vệ một tập đường dẫn nào đó, ví dụ

app.get('/secure/\*', coreCtrl.requireLogin);

Ví dụ như với mọi đường dẫn bắt đầu bằng /secure/ thì phải yêu cầu login.

Biểu thức chính quy thì mình hay dùng để validate (hợp thức hóa) các đối số, ví dụ

router.route('/author/:author([a-zA-Z0-9.\-\_]{8,30})').get(AuthorCtrl.renderAuthorPage);

Ở trên là một dạng đường dẫn với tham số :author được quy định có độ dài từ 8 đến 30 ký tự, bao gồm các chữ cái từ a-z, A-Z, 0-9 và 3 ký tự .,- và \_. Nếu ở vi phạm thì route sẽ không khớp.

Route parameters

Route parameters là những vị trí trên URL được đánh dấu bằng cách đặt tên, mục đích là để lấy ra các giá trị tương ứng. Tất cả cá giá trị đối số sẽ được đặt vào đối tượng req trong thuộc tính params. Với tên thuộc tính trùng khớp với từ khóa được xác định trên URL.

Ví dụ, chúng ta định nghĩa một path là /users/:user . Thì ở đây :user chính là một route param. Khi đó nếu người dùng truy cập đường dẫn như là

/users/nghuuquyen --> ta lấy ra được :user = nghuuquyen

và giá trị này sẽ nằm ở req.params.user

// Route path: /users/:user/:view

// Request URL: http://127.0.0.1:3000/users/nghuuquyen/gallery

// req.params: { "user": "nghuuquyen", "view": "gallery"}

app.route('/users/:user/:view', function(req, res) {

console.log(req.params.user);

console.log(req.params.view);

});

Ứng dụng

Route parameter dùng để biết được client muốn truy vấn cái gì thông qua đối số truyền vào. Dựa vào đó trong function handler tương ứng, chúng ta sẽ lấy các giá trị ra và thực hiện truy vấn phù hợp.

Ví dụ:

Path: /user/:user Request: /user/nghuuquyen

-> Trả về trang profile của người dùng có username là nghuuquyen.

Kỹ thuật nâng cao

* 1. Thường thì sẽ hay kết hợp rằng buộc biểu thức chính quy cho đối số để giảm bớt các lỗi truy vấn và tăng độ an toàn cho một route.

Ví dụ định nghĩa một route là /users/:user

với đối số :user mong muốn là một username. Lúc này mình biết rõ là username của ứng dụng có độ dài từ 8 đến 30 kí tự chỉ bao gồm chữ cái tiếng anh hoa, thường , chữ số từ 0 đến 9 và ba ký tự đặc biệt là -, . và \_.

Như vậy có thể áp dụng một biểu thức chính quy cho đối số này

:user(^[A-Za-z0-9.-\_]{8,30}+$) ==> Lúc này nếu bạn nhập vào một đường dẫn có đối số :user nhỏ hơn 8 hoặc lớn 30 ký tự thì route trên sẽ không nhận, hoặc chứa ký tự khác dấu ., \_ và - , thì route cũng không nhận. Từ đó hạn chế được việc query một username bị sau quy tắc và lại an toàn tránh tấn công SQL injection.

* 1. router.param(name, callback) là một kỹ thuật cho phép gán các hàm xử lý cho một đối số cụ thể trên route.

router.route('/:author([a-zA-Z0-9.\-\_]{8,30})')

.get(AuthorCtrl.renderAuthorPage);

Thì chúng ta sẽ áp dụng kỹ thuật trên để gán một hàm xử lý vào param này.

// Resolve route params

router.param('author', AuthorCtrl.findOne);

Nghĩa là cữ mỗi khi gặp route có param :author thì sẽ gọi hàm AuthorCtrl.findOne

Ứng dụng

Thay vì ở mỗi controller có đối số :author phải gọi lại service để lấy author thì như trên chỉ cần viết một lần là đủ, rất tiện và tránh trùng lặp code.

Route handlers

Đơn giản là một hoặc nhiều function sẽ được gọi khi một route trùng khớp để đáp ứng một yêu cầu nào đó. Lưu ý các handler sẽ được gọi đúng theo thứ tự truyền vào. Ví dụ

app.get('/user', [a, b]);

thì a sẽ gọi trước b. Chú ý là để b được gọi thì trong a phải gọi hàm next(). Ví dụ

var cb0 = function (req, res, next) {

console.log('CB0')

next()

}

var cb1 = function (req, res, next) {

console.log('CB1')

next()

}

var cb2 = function (req, res) {

res.send('Hello from C!')

}

app.get('/c', [cb0, cb1, cb2]);

function isLogged(req, res, next) {

if( Đã login ) {

return next();

}

return res.status(403).redirect('/login');

}

function renderProfilePage(req, res, next) {

// Do render profile page...

}

app.get('/users/:user', [isLogged, renderProfilePage]);

Hoặc thực hiện việc logging thông tin các request để lưu trữ xem ai làm gì, ví dụ:

function isLogged(req, res, next) {

if( Đã login ) {

return next();

}

return res.status(403).redirect('/login');

}

function renderProfilePage(req, res, next) {

// Do render profile page...

}

function logRequest(req, res, next) {

// Do logging IP, Header request

next();

}

app.get('/users/:user', [isLogged, logRequest, renderProfilePage]);

Response methods

Sau việc tiếp nhận và xử lý, thì việc tiếp theo đó là đáp ứng (Response). Trong express định nghĩa sẵn một số phương thức hỗ trợ cho bạn như là:

Tên phương thức Ý nghĩa

res.json() Trả về một dữ liệu dạng JSON

res.redirect() Chuyển hướng đến đường dẫn nào đó

res.render() Trả về một view template

res.send() gửi dữ liệu dạng text

REST APIs and Json

REST là từ viết tắt của Representational State Transfer. Đó là kiến trúc tiêu chuẩn web và Giao thức HTTP, là một kiểu kiến trúc lập trình, nó định nghĩa các quy tắc để thiết kết các web service chú trọng vào tài nguyên hệ thống. Trong kiến trúc REST mọi thứ đều được coi là tài nguyên, chúng có thể là: tệp văn bản, ảnh, trang html, video, hoặc dữ liệu động… REST server cung cấp quyền truy cập vào các tài nguyên, REST client truy cập và thay đổi các tài nguyên đó. Ở đây các tài nguyên được định danh dựa vào URI, REST sử dụng một vài đại diện để biểu diễn các tài nguyên như văn bản, JSON, XML. REST yêu cầu các nhà phát triển sử dụng một các rõ ràng các phương thức HTTP và nhất quán với cách mà chúng được định nghĩa. Quy tắc này của REST giúp thiết lập ánh xạ một một giữa các hành động tạo, đọc, cập nhật và xóa với các phương thức HTTP. Theo đó sẽ có:

Để tạo một tài nguyên trên server ta dùng phương thức POST.

Để lấy(đọc) tài nguyên trên server ta dùng phương thức GET.

Để update tài nguyên trên server ta dùng phương thức PUT.

Để xóa tài nguyển trên server ta dùng phương thức DELETE.

Các ứng dụng RESTful sử dụng các yêu cầu HTTP để thực hiện bốn hoạt động được gọi là CRUD (C: Create, R: Read, U: Update, và D: Delete).

API là viết tắt của Application Programming Interface, phương thức kết nối với các thư viện và ứng dụng khác.

Json:

JSON là một kiểu định dạng dữ liệu trong đó sử dụng văn bản thuần tuý, định dạng JSON sử dụng các cặp key - value để dữ liệu sử dụng .

Một ví dụ một tập tin có tên application\_info.json với nội dung như ở dưới đây sử dụng format kiểu JSON để lưu trữ thông tin của một ứng dụng phần mềm:

{

"name" : "PHPStorm",

"version" : "16.0.1",

"license" : "commercial"

}

Tập tin json có thể được lưu với bất kỳ phần mở rộng nào, tuy nhiên thông thường thì nó được lưu dưới phần mở rộng là .json hoặc .js.

JSON ban đầu được phát triển để dành phục vụ cho ứng dụng viết bằng JavaScript. Bản thân thuật ngữ JSON là viết tắt của cụm từ JavaScript Object Notation . Tuy nhiên vì JSON là một định dạng dữ liệu nên nó có thể được sử dụng bởi bất cứ ngôn ngữ nào àm không giới hạn với JavaScript.

Như vậy cú pháp của JSON rất đơn giản là mỗi thông tin dữ liệu sẽ có 2 phần đó là key và value, điều này tương ứng trong CSDL là tên field và giá trị của nó ở một record nào đó. Tuy nhiên nhìn qua thì đơn giản nhưng nếu chúng ta mổ xẻ nó ra thì có một vài điều như sau:

Chuỗi JSON được bao lại bởi dấu ngoặc nhọn {}

Các key, value của JSON bắt buộc phải đặt trong dấu nháy kép {"}, nếu bạn đặt nó trong dấu nháy đơn thì đây không phải là một chuỗi JSON đúng chuẩn.

Nếu có nhiều dữ liệu (nhiều cặp key => value) thì ta dùng dấu phẩy (,) để ngăn cách

Các key của JSON bạn nên đặt chữ cái không dấu hoặc số, dấu \_ và không có khoảng trắng., ký tự đầu tiên không nên đặt là số.

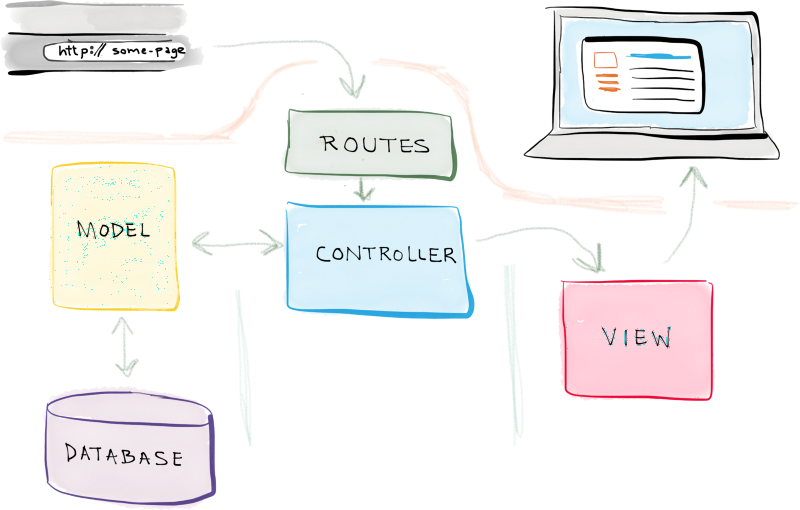
MVC in Express:

Mô hình MVC là gì: Là một kiến trúc phần mềm hay mô hình thiết kế được sử dụng trong kỹ thuật phần mềm, là mô hình phân bố source code thành 3 phần, mỗi thành phần có một nhiệm vụ riêng biệt và độc lập với các thành phần khác.

M là Model: cấu trúc dữ liệu theo cách tin cậy và chuẩn bị dữ liệu theo lệnh của controller

V là View: Hiển thị dữ liệu cho người dùng theo cách dễ hiểu dựa trên hành động của người dùng.

C là Controller: Nhận lệnh từ người dùng, gửi lệnh đến cho Model để cập nhập dữ liệu, truyền lệnh đến View để cập nhập giao diện hiển thị.



Qui trình xử lí chung của web hiện đại thường như sau:

Người dùng gọi yêu cầu xử lí tại trang chủ.

“Controller” nhận yêu cầu này và đưa lệnh xử lí yêu cầu đó. Các lệnh thực thi với phần “View” thì cập nhập hoặc phục vụ yêu cầu trang web, với “Model” thì để trình diễn logic. Giả sử lệnh yêu cầu có yếu tố logic.

“Model” thực thi phần logic được lấy từ cơ sở dữ liệu và gửi trả lại phản hồi dựa trên hướng dẫn của “Controller”.

“Controller” truyền dữ liệu ra phần “View”, cập nhật giao diện hiển thị cho người dùng.

Mô hình MVC trong Express:

Thư mục views chứa các file template (file có phần mở rộng là .ejs), các file này được dùng để hiển thị dữ liệu, tức là tương tự với phần Views trong MVC.

Thư mục routes được dùng để chuyển hướng các URL đến các hàm xử lý tương ứng, tức là tương tự với Controller trong MVC.

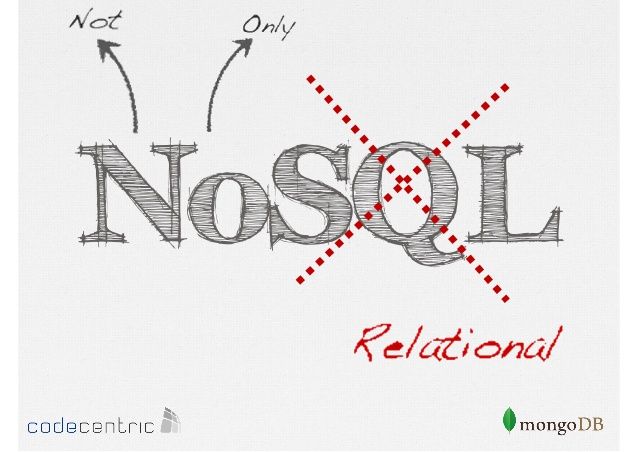
Thư mục models có chức năng lưu trữ dữ liệu, thay đổi/cập nhật dữ liệu, hỗ trợ truy vấn dữ liệu. Được lưu trong một thư mục riêng tách rời với views và routes. Đây cũng là nơi chứa các Schema của models

MongoDB:

Định nghĩa về MongoDB:

MongoDB là một cơ sở dữ liệu mã nguồn mở và là cơ sở dữ liệu NoSQL(\*) hàng đầu, được hàng triệu người sử dụng. MongoDB được viết bằng C++.

Ngoài ra, MongoDB là một cơ sở dữ liệu đa nền tảng, hoạt động trên các khái niệm Collection và Document, nó cung cấp hiệu suất cao, tính khả dụng cao và khả năng mở rộng dễ dàng.



(\*) NoSQL là 1 dạng CSDL mã nguồn mở không sử dụng Transact-SQL để truy vấn thông tin. NoSQL viết tắt bởi: None-Relational SQL, hay có nơi thường gọi là Not-Only SQL. CSDL này được phát triển trên Javascript Framework với kiểu dữ liệu JSON. (Cú pháp của JSON là “key:value”) NoSQL ra đời như là 1 mảnh vá cho những khuyết điểm và thiếu xót cũng như hạn chế của mô hình dữ liệu quan hệ RDBMS về tốc độ, tính năng, khả năng mở rộng, memory cache,...

Các thuật ngữ hay sử dụng trong MongoDB:

* **\_id** – Là trường bắt buộc có trong mỗi document. Trường \_id đại diện cho một giá trị duy nhất trong document MongoDB. Trường \_id cũng có thể được hiểu là khóa chính trong document. Nếu bạn thêm mới một document thì MongoDB sẽ tự động sinh ra một \_id đại diện cho document đó và là duy nhất trong cơ sở dữ liệu MongoDB.
* **Collection** – Là nhóm của nhiều document trong MongoDB. Collection có thể được hiểu là một bảng tương ứng trong cơ sở dữ liệu RDBMS (Relational Database Management System). Collection nằm trong một cơ sở dữ liệu duy nhất. Các collection không phải định nghĩa các cột, các hàng hay kiểu dữ liệu trước.
* **Cursor** – Đây là một con trỏ đến tập kết quả của một truy vấn. Máy khách có thể lặp qua một con trỏ để lấy kết quả.
* **Database** – Nơi chứa các Collection, giống với cơ sở dữ liệu RDMS chúng chứa các bảng. Mỗi Database có một tập tin riêng lưu trữ trên bộ nhớ vật lý. Một mấy chủ MongoDB có thể chứa nhiều Database.
* **Document** – Một bản ghi thuộc một Collection thì được gọi là một Document. Các Document lần lượt bao gồm các trường tên và giá trị.
* **Field** – Là một cặp name – value trong một document. Một document có thể có không hoặc nhiều trường. Các trường giống các cột ở cơ sở dữ liệu quan hệ.
* **JSON** – Viết tắt của JavaScript Object Notation. Con người có thể đọc được ở định dạng văn bản đơn giản thể hiện cho các dữ liệu có cấu trúc. Hiện tại JSON đang hỗ trợ rất nhiều ngôn ngữ lập trình.
* **Index** – Là những cấu trúc dữ liệu đặc biệt, dùng để chứa một phần nhỏ của các tập dữ liệu một cách dễ dàng để quét. Chỉ số lưu trữ giá trị của một fields cụ thể hoặc thiết lập các fields, sắp xếp theo giá trị của các fields này. Index hỗ trợ độ phân tích một cách hiệu quả các truy vấn. Nếu không có chỉ mục, MongoDB sẽ phải quét tất cả các documents của collection để chọn ra những document phù hợp với câu truy vấn. Quá trình quét này là không hiệu quả và yêu cầu MongoDB để xử lý một khối lượng lớn dữ liệu.

Hãy lưu ý sự khác biệt của các trường và \_id trong một document. Một \_id được dùng để đại diện cho một document và chúng được sinh ra khi thêm một Document vào Collection.

Các kiểu dữ liệu trong MongoDB:

MongoDB hỗ trợ các kiểu dữ liệu sau:

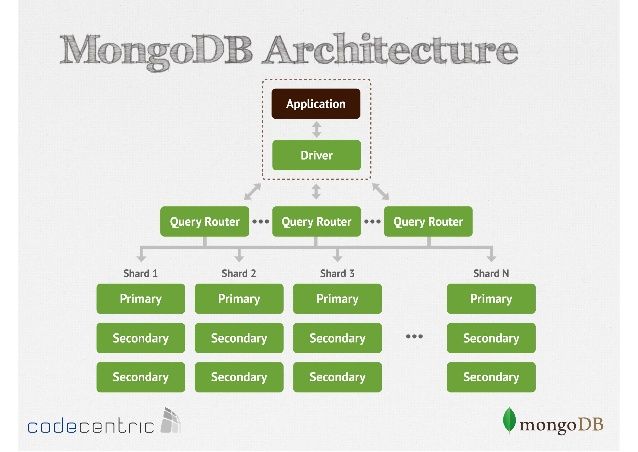
| **Type** | **Number** | **Alias** |
| --- | --- | --- |
| Double | 1 | “double” |
| String | 2 | “string” |
| Object | 3 | “object” |
| Array | 4 | “array” |
| Binary data | 5 | “binData” |
| Undefined | 6 | “undefined” |
| ObjectId | 7 | “objectId” |
| Boolean | 8 | “bool” |
| Date | 9 | “date” |
| Null | 10 | “null” |
| Regular Expression | 11 | “regex” |
| DBPointer | 12 | “dbPointer” |
| JavaScript | 13 | “javascript” |
| Symbol | 14 | “symbol” |
| JavaScript (with scope) | 15 | “javascriptWithScope” |
| 32-bit integer | 16 | “int” |
| Timestamp | 17 | “timestamp” |
| 64-bit integer | 18 | “long” |
| Decimal128 | 19 | “decimal” |
| Min key | -1 | “minKey” |
| Max key | 127 | “maxKey” |

* **Chuỗi:** Đây là kiểu dữ liệu được sử dụng phổ biến nhất để lưu giữ dữ liệu. Chuỗi trong MongoDB phải là UTF-8 hợp lệ.
* **Số nguyên:** Kiểu dữ liệu này được sử dụng để lưu một giá trị số. Số nguyên có thể là 32 bit hoặc 64 bit phụ thuộc vào Server của bạn.
* **Boolean**: Kiểu dữ liệu này được sử dụng để lưu giữ một giá trị Boolean (true/false).
* **Double**: Kiểu dữ liệu này được sử dụng để lưu các giá trị số thực dấu chấm động.
* **Min/ Max keys:** Kiểu dữ liệu này được sử dụng để so sánh một giá trị với các phần tử BSON thấp nhất và cao nhất.
* **Mảng:** Kiểu dữ liệu này được sử dụng để lưu giữ các mảng hoặc danh sách hoặc nhiều giá trị vào trong một key.
* **Timestamp:** Đánh dấu thời điểm một Document được sửa đổi hoặc được thêm vào.
* **Object:** Kiểu dữ liệu này được sử dụng cho các Document được nhúng vào.
* **Null:** Kiểu dữ liệu này được sử dụng để lưu một giá trị Null.
* **Symbol:** Kiểu dữ liệu này được sử dụng giống như một chuỗi
* **Date :** Kiểu dữ liệu này được sử dụng để lưu giữ date và time hiện tại trong định dạng UNIX time.
* **Object ID:** Kiểu dữ liệu này được sử dụng để lưu giữ ID của Document.
* **Binary data:** Kiểu dữ liệu này được sử dụng để lưu giữ dữ liệu nhị phân.
* **Code:** Kiểu dữ liệu này được sử dụng để lưu giữ JavaScrip code vào trong Document.
* **Regular expression:** Kiểu dữ liệu này được sử dụng để lưu giữ Regular Expresion.

*Ví dụ :* Dữ liệu của collection bao gồm 2 documents MinKey và MaxKey: { "\_id" : 1, x : { "$minKey" : 1 } } { "\_id" : 2, y : { "$maxKey" : 1 } }

Một số câu lệnh trong MongoDB:

| **Câu lệnh** | **SQL** | **MongoDB** |
| --- | --- | --- |
| Create table | CREATE TABLE people (id MEDIUMINT NOT NULL AUTO\_INCREMENT, user\_id Varchar(30), age Number, status char(1), PRIMARY KEY (id)) | db.people.insertOne({User\_id: “abc123”, Age: 55, Status: “A”}) |
| Drop table | DROP TABLE people | db.people.drop() |
| Insert records into tables | INSERT INTO people(user\_id, age, status) VALUES ("bcd001", 45, "A") | db.people.insertOne( { user\_id: "bcd001", age: 45, status: "A" }) |
| Select | SELECT \*FROM people | db.people.find() |
|  | SELECT id,user\_id, status FROM people | db.people.find( { }, { user\_id: 1, status: 1 } ) |
|  | SELECT \* FROM people WHERE status = "A" | db.people.find( { status: "A" } ) |
|  | SELECT \* FROM people WHERE status = "A" AND age = 50 | db.people.find( { status: "A", age: 50 } ) |
|  | SELECT \* FROM people WHERE status = "A" OR age = 50 | db.people.find( { $or: [ { status: "A" } , { age: 50 } ] } ) |
|  | SELECT \* FROM people WHERE user\_id like "%bc%" | db.people.find( { user\_id: /bc/ } )  db.people.find( { user\_id: { $regex: /bc/ } } ) |
|  | SELECT COUNT(user\_id) FROM people | db.people.count( { user\_id: { $exists: true } } )  db.people.find( { user\_id: { $exists: true } } ).count() |
| Update records | UPDATE people SET status = "C" WHERE age > 25 | db.people.updateMany( { age: { $gt: 25 } }, { $set: { status: "C" } } ) |
|  | UPDATE people SET age = age + 3 WHERE status = "A" | db.people.updateMany( { status: "A" } , { $inc: { age: 3 } } ) |
| Delete Records | DELETE FROM people WHERE status = "D" | db.people.deleteMany( { status: "D" } ) |
|  | DELETE FROM people | db.people.deleteMany({}) |

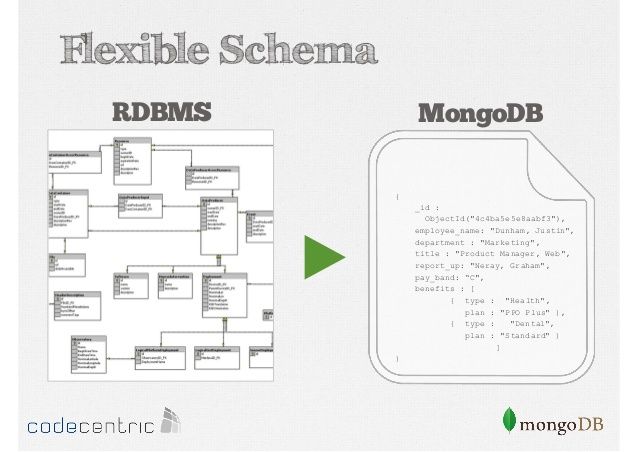
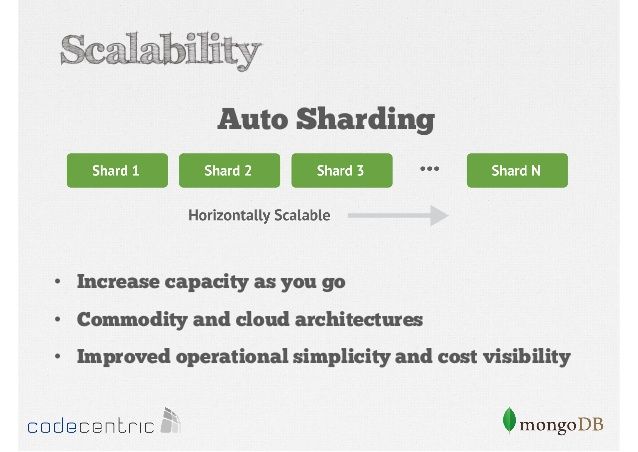
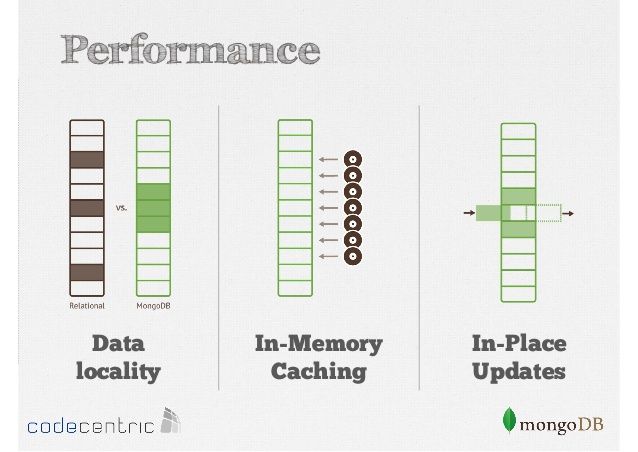
MongoDB hoạt động như thế nào:   


* MongoDB hoạt động dưới một tiến trình ngầm service, luôn mở một cổng (Cổng mặc định là 27017) để lắng nghe các yêu cầu truy vấn, thao tác từ các ứng dụng gửi vào sau đó mới tiến hành xử lý.
* Mỗi một bản ghi của MongoDB được tự động gắn thêm một field có tên “\_id” thuộc kiểu dữ liệu ObjectId mà nó quy định để xác định được tính duy nhất của bản ghi này so với bản ghi khác, cũng như phục vụ các thao tác tìm kiếm và truy vấn thông tin về sau. Trường dữ liệu “\_id” luôn được tự động đánh index (chỉ mục) để tốc độ truy vấn thông tin đạt hiệu suất cao nhất.
* Mỗi khi có một truy vấn dữ liệu, bản ghi được cache (ghi đệm) lên bộ nhớ Ram, để phục vụ lượt truy vấn sau diễn ra nhanh hơn mà không cần phải đọc từ ổ cứng.
* Khi có yêu cầu thêm/sửa/xóa bản ghi, để đảm bảo hiệu suất của ứng dụng mặc định MongoDB sẽ chưa cập nhật xuống ổ cứng ngay, mà sau 60 giây MongoDB mới thực hiện ghi toàn bộ dữ liệu thay đổi từ RAM xuống ổ cứng.

**Từ đây có thể nhìn thấy *nhược điểm* của Mongodb như sau:**

* Dữ liệu được caching, lấy RAM làm trọng tâm hoạt động vì vậy khi hoạt động yêu cầu một bộ nhớ RAM lớn
* Như đã giới thiệu ở trên, mọi thay đổi về dữ liệu mặc định đều chưa được ghi xuống ổ cứng ngay lập tức vì vậy khả năng bị mất dữ liệu từ nguyên nhân mất điện đột xuất là rất cao.

Lợi thế của MongoDB:

* Ít schema hơn: Vì schema được sinh ra là để nhóm các đối tượng vào 1 cụm, dễ quản lý. Ví dụ như tạo 1 schema tên là Students chẳng hạn thì chỉ có những gì liên quan đến student thì mới được cho vào schema này. Trong khi đó trong mongodb thì chỉ 1 collection ta có thể chứa nhiều document khác nhau . Với mỗi document thì số trường, nội dung, kích thước lại có thể khác nhau. 
* Cấu trúc của một đối tượng rõ ràng.
* Không có các Join phức tạp.
* Khả năng mở rộng cực lớn: việc mở rộng dữ liệu mà không phải lo đến các vấn đề như khóa ngoại, khóa chính, kiểm tra ràng buộc, ... MongoDB cho phép thực hiện replication và sharding nên việc mở rộng cũng thuận lợi hơn. 
* Sử dụng bộ nhớ trong để lưu giữ cửa sổ làm việc cho phép truy cập dữ liệu nhanh hơn. Việc cập nhật được thực hiện nhanh gọn nhờ update tại chỗ (in-place). 

Khi nào nên sử dụng MongoDB:

Sử dụng MongoDB trong trường hợp:

* Nếu website của bạn có tính chất INSERT cao Bởi vì mặc định MongoDB có sẵn cơ chế ghi với tốc độ cao và an toàn.Website của bạn ở dạng thời gian thực nhiều, nghĩa là nhiều người thao tác với ứng dung. Nếu trong quá trình load bị lỗi tại một điểm nào đó thì nó sẽ bỏ qua phần đó nên sẽ an toàn.
* Website bạn có nhiều dữ liệu quá Giả sử web bạn có đến 10 triệu records thì đó là cơn ác mộng với MYSQL. Bởi vì MongoDB có khả năng tìm kiến thông tin liên quan cũng khá nhanh nên trường hợp này nên dùng nó.
* Máy chủ không có hệ quản trị CSDL Trường hợp này thường bạn sẽ sử dụng SQLITE hoặc là MongoDB.

Khi nào không nên sử dụng MongoDB:

* Các ứng dụng cần sử dụng nhiều transaction (như ngân hàng) do Mongodb không có cơ chế transaction (giao dịch) để phục vụ cho các ứng dụng ngân hàng
* Các ứng dụng cần SQL (sử dụng joins).

Các công cụ quản trị MongoDB:

Một số công cụ điển hình như:

* RoboMongo
* UMongo(trước đây là JMongoBrower)
* MongoExplorer
* RockMongo
* MongoDB Compass Community